Αναφορά 1ης Εργαστηριακής Άσκησης στο μάθημα
Δομές Δεδομένων & Αρχείων
Καπελώνης Χαρίλαος, ΑΜ: 2020030058

1) Υλοποίηση ζητούμενων δομών δεδομένων και αρχείων σε Java:

Λίγα λόγια για τον κώδικα:
Το πρόγραμμα υλοποιεί τις ζητούμενες δομές, δύο στη μνήμη (A1, A2) και δύο στον δίσκο (B1, B2). Οι αντίστοιχες κλάσεις βρίσκονται στα πακέτα memoryStructures και diskStructures. Το πακέτο myPackage αφορά κλάσεις βοηθητικές ή/και κοινές για τα δύο άλλα πακέτα. Επίσης έχει την κλάση Main και την κλάση Tester. Τέλος, αναφέρουμε ότι τα δεδομένα (x, y) αναπαριστώνται με στιγμιότυπα της κλάσης Point.
Στις δομές του δίσκου, αυτό που διαφοροποιείται είναι ότι τα δεδομένα (Points) που εισάγουμε αποθηκεύονται σε ένα αρχείο, αλλά με συγκεκριμένο τρόπο. Αρχικά, από το Point (x, y) μετατρέπουμε το x και το y σε bytes (8 bytes) και τα προσθέτουμε σε έναν πίνακα με μέγεθος 256 bytes. Άρα, μπορούμε να αποθηκεύσουμε 32 Points σε αυτόν τον byte πίνακα που, επί της ουσίας, *είναι* τα αντικείμενα της κλάσης DataPage. Μια σελίδα δίσκου DataPage, λοιπόν, μπορεί να αποθηκευτεί τώρα σε αρχείο. Αυτό γίνεται με την βοήθεια της κλάσης FileController. Τέλος, μπορούμε και να διαβάζουμε από το αρχείο, DataPages.

A1. class MemList
Για τη συνδεδεμένη λίστα στην μνήμη, αρχικά, κατασκευάσαμε την κλάση MemListItem που τα fields του είναι ένα Point και ένα reference σε ένα άλλο MemListItem. Η κλάση MemList έχει ένα head, reference στο πρώτο στοιχείο που εισήχθη στη λίστα κι ένα tail για το πιο πρόσφατο. Κάνουμε insert σε αυτήν τη δομή δημιουργώντας ένα αντικείμενο item και κρατάμε αναφορά σε αυτό με το tail. Όσον αφορά την αναζήτηση, ψάχνουμε σειριακά μέσα στη λίστα κάνοντας χρήση head και tail.

A2. class MemHash
Το νόημα αυτής της δομής είναι ότι δημιουργούμε έναν πίνακα M size που κάθε στοιχείο του είναι ένα MemList. Όταν θέλουμε να κάνουμε insert ένα Point (x, y), περνάμε τα (x, y) από μία συνάρτηση Hash που μας γυρνάει αριθμό από 0 έως M-1, και προσθέτουμε το Point στην αντίστοιχη λίστα ανάλογα (με την insert της MemList). Για την αναζήτηση εργαζόμαστε ανάλογα: πάλι μέσω της Hash βρίσκουμε το index στον πίνακα και αναζητούμε με την συνάρτηση της MemList.

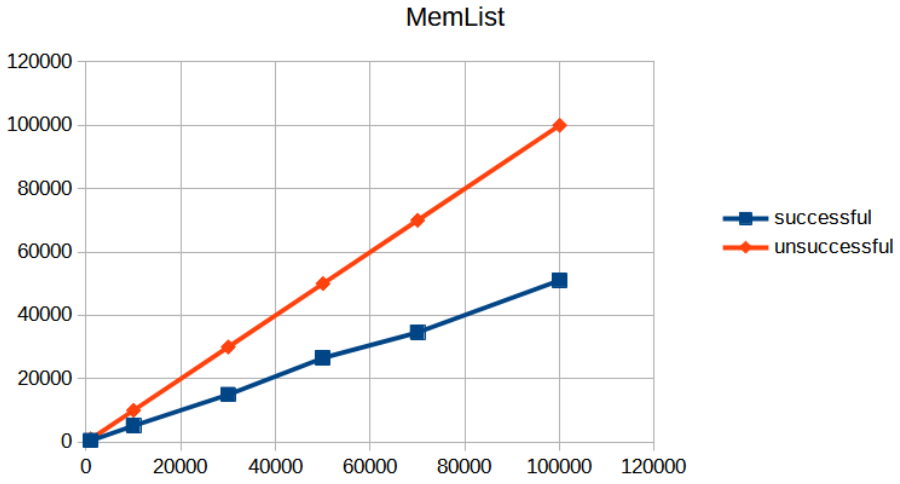
B1. class DiskPageStruct
Όταν γεμίζει μια σελίδα δίσκου DataPage αποθηκεύεται στο αρχείο. Αυτό υλοποιείται με την κλάση DiskPageStruct και απαντάει στο ερώτημα B1. Σχετικά με την αναζήτηση, αυτό που κάνουμε είναι να διαβάζουμε μια σελίδα από τον δίσκο, να ελέγχουμε επί της τρέχουσας σελίδας αν υπάρχει το στοιχείο κι αυτή η διαδικασία να γίνεται επαναληπτικά μέχρι είτε να βρεθεί το στοιχείο είτε να φτάσουμε στο τέλος του αρχείου.

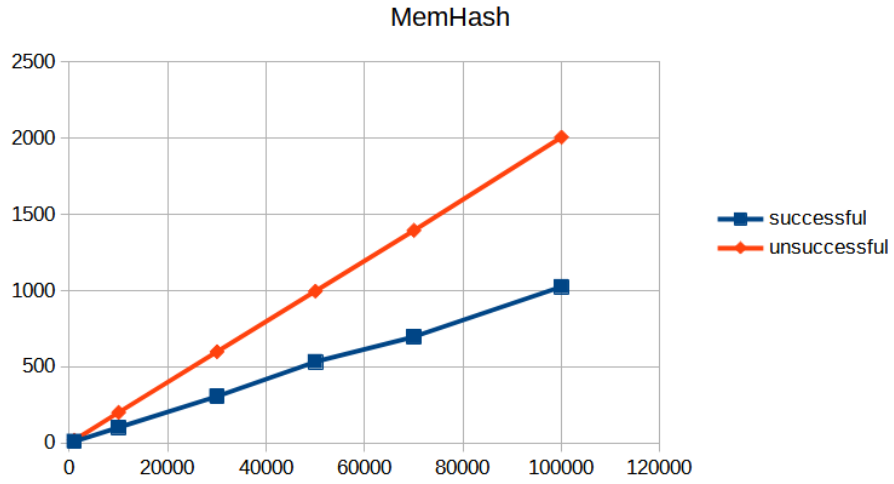
B2. class DiskHash
Η DiskHash αποτελείται από έναν πίνακα που έχει ως στοιχεία του DiskLists. Η DiskList είναι μία κλάση για συνδεδεμένη λίστα που, όμως, τα στοιχεία του δεν είναι Points, αλλά DataPages. Συνεπώς,

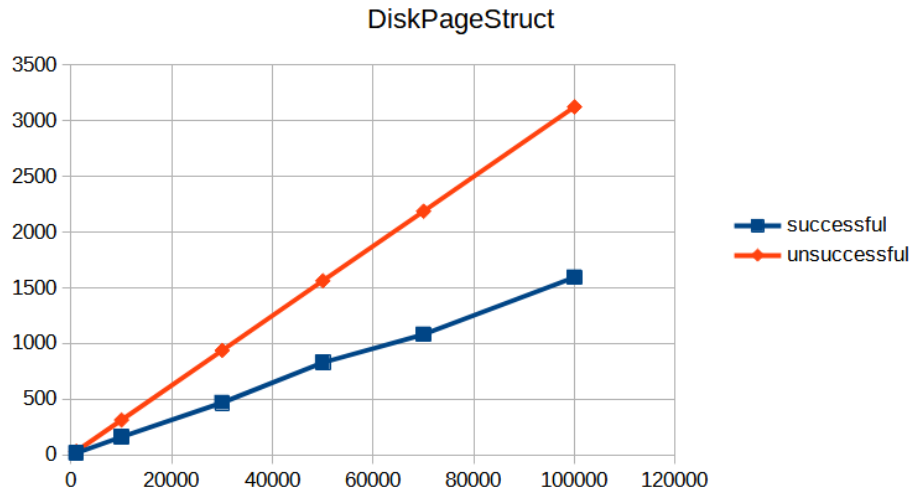
έχουμε ίδια λογική με την MemHash. Η διαφορά έγκειται στο γεγονός ότι οι λίστες αποθηκεύονται στο αρχείο. Έτσι κάθε DiskListItem, εκτός από το reference στο DataPage του, έχει και ένα field που είναι long και λέγεται address: είναι η διεύθυνση της DataPage - η οποία είναι next - στο αρχείο. Η αναζήτηση γίνεται με διαδοχικό διάβασμα των DataPages που αντιστοιχούν σε μία DiskList. Επί της σελίδας δίσκου που έχει διαβαστεί γίνεται αναζήτηση.

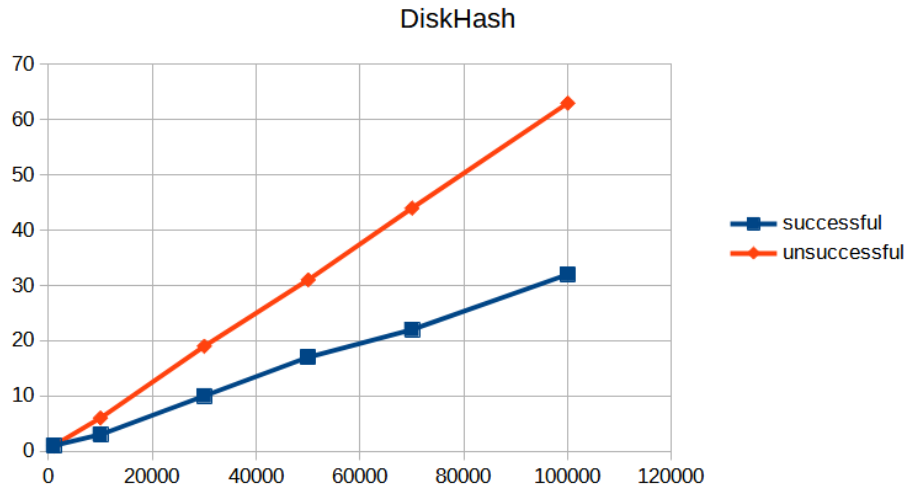
2) Διαγράμματα:

Τρέξαμε τον κώδικα και με τη βοήθεια της κλάσης Tester κάναμε τα ζητούμενα τεστ, δηλαδή για προσθήκη 1.000, 10.000, 30.000, 50.000, 70.000 και 100.000 Points, κάναμε 100 επιτυχείς (υπάρχοντα Points – μπλε καμπύλη) και 100 ανεπιτυχείς (πορτοκαλί καμπύλη):









3) Ανάλυση κάθε μεθόδου ως προς την πολυπλοκότητα:

Για το A1, η πολυπλοκότητα αυξάνεται γραμμικά. Αναζητάμε ένα στοιχείο επί της λίστας, όμως αναγκαστικά πρέπει να διασχίσουμε όλη τη λίστα. Οπότε, όσο περισσότερα δεδομένα έχουμε, τόσο περισσότερες συγκρίσεις γίνονται, και μάλιστα πολλές. Για το A2, η πολυπλοκότητα αυξάνεται γραμμικά, αλλά με μικρότερο ρυθμό από το A1. Αυτό επεξηγείται όμορφα με νούμερα: έστω ότι έχουμε ομοιόμορφη κατανομή πιθανότητας ένα Point να εισαχθεί στη λίστα του πίνακα της MemHash μετά από το πέρασμα του από την συνάρτηση hash, δηλαδή έστω ότι έχουμε πίνακα 50 στοιχείων (λίστες) και 5.000 δεδομένα εισαχθέντα. Κάθε λίστα από τις 50 θα έχει 5.000/50=100 Points. Έστω ότι αναζητάμε ένα Point που δεν υπάρχει, τότε θα γίνουν 100 συγκρίσεις, εφόσον γλιτώνουμε τις υπόλοιπες 4.900 συγκρίσεις που θα γίνονταν στη δομή του A1. Αυτό οφείλεται σε ένα πρώτο “ξεκαθάρισμα” που κάνει η hash function. Όμοια πράγματα μπορούμε να πούμε και για τα B1, B2 ερωτήματα με τη διαφορά ότι disk access (των B ερωτημάτων) = 32 συγκρίσεις (των A ερωτημάτων). Οπότε τα αποτελέσματα, σχεδόν ομοιάζουν.

4) Πώς θα αποθηκεύαμε τον πίνακα της DiskHash σε αρχείο:

Σε αυτό το πρόγραμμα κάναμε το δικό μας serialize και δεν χρησιμοποιήσαμε αυτό της Java. Για το ερώτημα αυτό όμως χρειάζεται να κάνουμε Object Serialization, και μετά τα δεδομένα να τα γράφουμε σε ένα άλλο αρχείο.

5) Πηγές - Συνάδελφοι:

https://stackoverflow.com/questions/1994255/how-to-write-console-output-to-a-txt-file
https://stackoverflow.com/questions/2183240/java-integer-to-byte-array
Με τον συνάδελφο Χρήστο Κωσταδήμα ανταλλάξαμε κάποιες σκέψεις και αποτελέσματα με συνεργατικό χαρακτήρα, με σκοπό ο ένας να υποδείξει κάποιο λάθος στον άλλον, και όχι με σκοπό την αντιγραφή.